Material complementario 12/05/2020

Arno Formella

Este documento recoge más o menos lo que hubiese contado en clases presenciales a lo largo de la exposición paulatina de las transparencias. En clase normalmente descubro diferentes partes del contenido de cada transparencia poco a poco, para no sobrecargar con información de golpe, y para mantener un hilo de pensamiento. En las transparencias distribuidas todo viene de golpe, por eso recomiendo trabajar con ellas lentamente y ver este material complementario a la par.

Estarán incluidas preguntas (a veces sin respuestas) para animar a la reflexión sobre el tema y a la búsqueda de información adicional. Admás proporciono más referencias a la red y la bibliografía.

Ojo, las páginas mencionadas aquí siempre se refieren a los tochos que publico para cada semana. Puede ser (y es casi seguro) que en el documento global (que desarrollo en paralelo) la enumeración de las página va a variar, ya que se añaden transparencias en otros lugares.

P284

Esta división en dos clases de tipos de propiedades de un programa concurrente viene dado por razones bastante obvias: si un programa no es *seguro* significa que puede ser que en algún momento ejecuta instrucciones que no debe, si un programa no es *vivo* significa que puede ser que en algún momento no ejecuta instrucciones que debe.

Normalmente problemas de seguridad se deben eliminar antes de distribuir el programa (o en un plan de garantías al producto software), problemas de vivacidad se pueden arreglar con *parches* una vez en el cliente (o cobrar para las nuevas versiones arregladas :-)

P285

Son tipicos ejemplos de posibles problems de seguridad.

Recuerda que dentro del ámbito de sincronización está el problema de las condiciones de carrera.

P286

La inanición (temporal) de un proceso no necesariamente tiene que convertirse en un problema de seguridad ya que pueda ser que el trabajo por realizar se encarga a otro proceso, pero puede ser que se pierda eficiencia.

P287

El problema típico que suele conllevar la pérdida de eficiencia es el problema de la espera activa, es decir, que un proceso de forma activa está comprobando (y con eso consumiendo ciclos de CPU) si se cumple una condición para reanudar su trabajo. Mucho más eficiente es que tal proceso se mantiene en espera y se le despierta cuando la condición se cumpla.

El típico problema para el desarrollo de la aplicación es decidir cual será el proceso que despierta y cuando lo hace. O en términos de Java: quien hace cuando el `notify()` adecuado. Un simple `notifyAll()` en muchas ocaciones es una pérdida de eficiencia.

P288

Ya vimos en los protocolos de entrada y salida a las secciones críticas tal situación que hay que decidir cual es el siguiente proceso que puede entrar si varios procesos lo intentan a la vez y que repercusión tiene en una secuencia de intentos. (Recuerda que en el protocolo asimétrico, por ejemplo, uno de los procesos participantes puede llevar el otro proceso a inanición.)

Estos cuatro *ejemplos* son para daros unas ideas.

Los primeros dos son justicias que se pueden usar durante el analisis de todo una aplicación para argumentar, por ejemplo, la corrección del algoritmo concurrente.

Los últimos dos son ejemplos tipicamente encontradas en situaciones reales.

Pero hay muchas más posibilidades de implementar diferentes tipos de justicias. Para el diseñador de una aplicación concurrente es importante que sepa cual es la justicia disponible en las herramientas que está usando y cuales es la justicia que debe implementar/usar según los requisitos por cumplir.

P289

Si has usado una vez una aplicación en un dispositivo y observaste que la interfaz al usuario no responde (por ejemplo para terminar una aplicación o uno proceso de la misma) entonces, en mi opinión, está mal diseñada: si doy a abortar (o un botón) quiero que termina (reacciona) inmediatamente.

P290

Son unos detalles sobre la espera activa.

Existen casos donde la espera activa es inevitable. ¿Se te ocurre alguno?

P291

Es una técnica muy usada: quien libera un recurso da acceso al siguente, todos esperan en una cola.

Todos conocemos estas colas de la vida cotidiana.

En ciertas aplicaciones puede existir la necesidad de cambiar el orden del acceso por razones de prioridad, pero hay que tener cuidado en implementar también una justicia adecuada entre diferentes niveles de prioridad (para que no queden en inanición procesos de baja prioridad).

P292

Pues, son las razones porque dos (o más) procesos se tienen que comunicar.

P293

Estos trés métodos hemos visto en prácticas.

Es tarea durante el análisis y diseño de una aplicación concurrente incluir este aspecto: como realizar la comunicación entre los procesos.

Observa que en sistemas modernos incluso se puede pasar mensajes dentro del mismo ordenador (recuerda tus clases de sistemas operativos), pero también existen entornos de ordenadores distribuidos que emulan una memoria compartida (que puede simplificar el desarrollo de aplicaciones concurrentes). La mayoria son de investagación, pero seguramente en el futuro habrá nuevos intentos. Imagínate una máquina virtual de java que se ejecuta automáticamente de forma concurrente un diferentes dispositivos...

P294

Estos dos tipos de comunicación conocemos de diferentes medios que usamos.

P295

Básicamente son los dos tipos que se usan: canales basados en paquetes, y canales basados en flujos.

Mucho más aprendisteis en clases de redes.

P296

Son los cuatro fallos posibles. En cada aplicación se tiene que evaluar hasta que punto se puede tolerar tales fallos.

Si no se puede tolerar, hay que buscar medidas para que no ocurran dichos fallos.

P297

Son las medidas que normalmente se aplica (mira también clases de redes).

P298

A lo mejor no hais visto en clases de redes este tipo de comunicación donde no hacen falta asentimientos desde el receptor hacia el transmisor. (`https://en.wikipedia.org/wiki/Raptor_code`).

Se usan en redes móviles a partir de la generación 4.

P299

Son unos temas que quedan cortos en esta asignatura, los menciono para futuros estudios y para que tengáis la noción que existen y que sepáis que hay que tener las en cuenta cuando se desarrolla una aplicación concurrente.